
Microsoft®
PDC 2000
Professional Developers Conference
Microsoft®
.net™
the defining point

The .NET Framework, A COM Developer's Perspective

Dennis Angeline
Program Manager
.NET Frameworks
Common Language Runtime
Microsoft Corporation

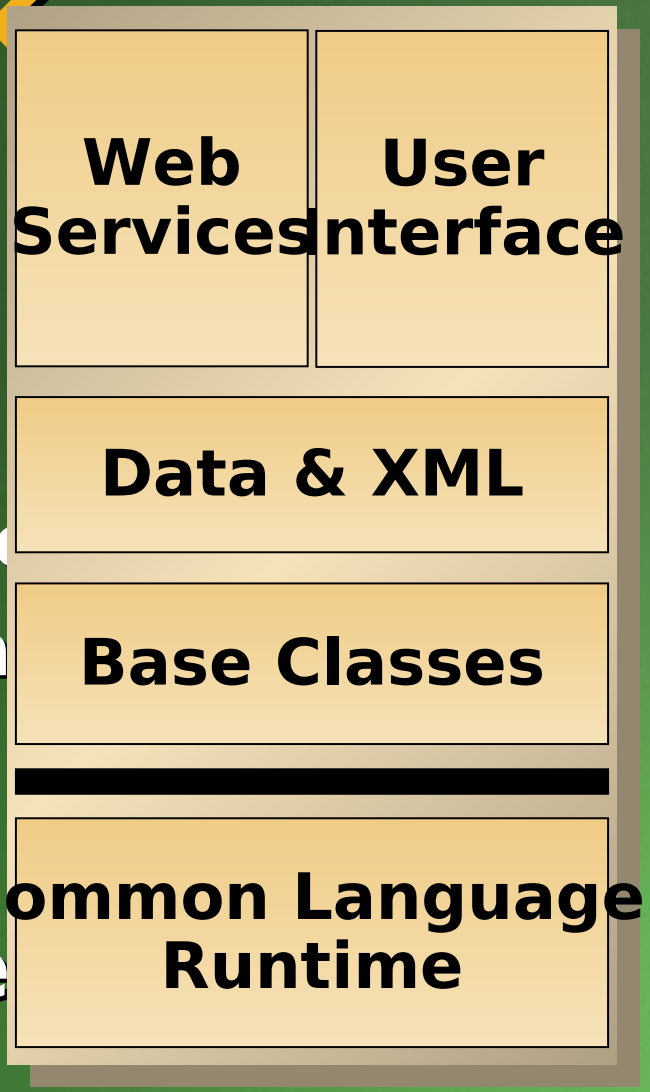
3-211

Agenda

- The Road To The .NET Framework
- It’s All In The Type System
- A Framework For Productivity
- Along With The Services You Need

The Road To The .NET Framework

- COM Themes
  - Code reusability
  - Language neutral
  - Component developme
  - Component marketpla
  - Preserve investment
- The .NET Framework continues to advance these COM themes



Common Language Runtime

- Productivity focused
  - Truly simplifies development
  - Eliminates the plumbing
  - Reduces the learning curve
  - Tightly integrates into languages
- Powerful new developer features
  - Implementation inheritance
  - Application isolation
  - Version resilient types
  - Evidence-based security

Productivity Focused

- Eliminates Boilerplate Code
  - No IUnknown
  - No IDispatch
  - No Class Factories
  - No Connection Points
- Simplifies Routine Tasks
  - No Reference Counting
  - No QueryInterface
  - No Registry updates
  - No HRESULTS or Guids
  - No Variants, BSTRS, SafeArrays

It’s All In The Type System

The Runtime Type System

- Integrated into compilers
  - For tighter language integration
  - Open to any compiler vendor
- Provides rich type definitions
  - Way beyond just interfaces
- Provides common type representation
  - Easy data transfer across languages
  - Common object representation across languages
- Allows intra-component interaction
  - As well as inter-component

Type System Improvements

- Classes, not just interfaces !!
  - With methods, properties, fields and events
  - Constructors (Parameterized)
  - Method Overloading
  - Static, instance and virtual methods
  - Public/private/assembly/family accessibility
  - Implementation Inheritance
- Value Types
  - Stack based value
  - Highly efficient
  - Object Semantics

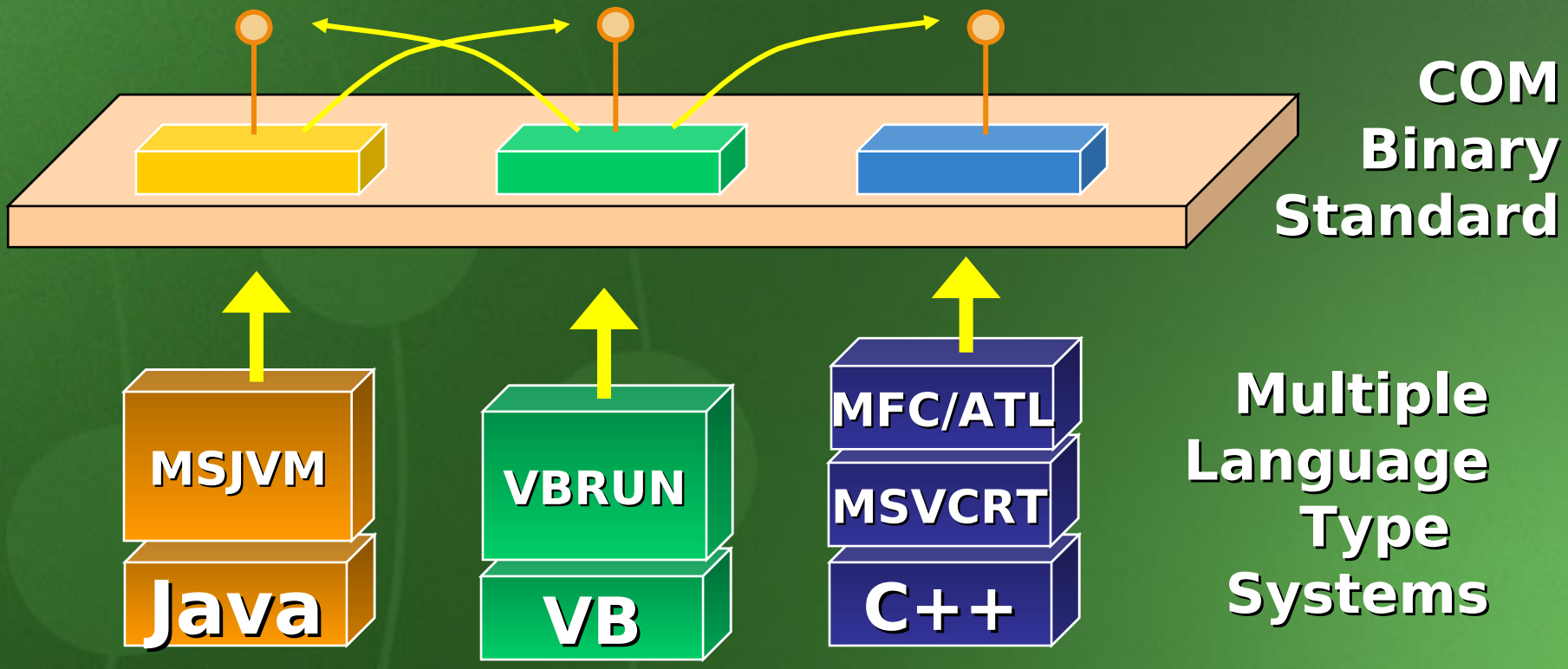
Implementation Inheritance

- Classes support implementation inheritance
  - Even across languages!
  - Single class, multiple interface
- Build libraries of reusable code
  - Can be used by any language
  - Can be extended by any language
  - Use common libraries across team using different tools
- For Example
  - .NET Frameworks provide complete winforms/webforms/data access/etc library
  - Third parties can build additional frameworks and specialized libraries $$$
  - Corporations can build business specific libraries

Reusable libraries

The COM Binary Standard

COM provided a standard for producing components that were *Binary Compatible*

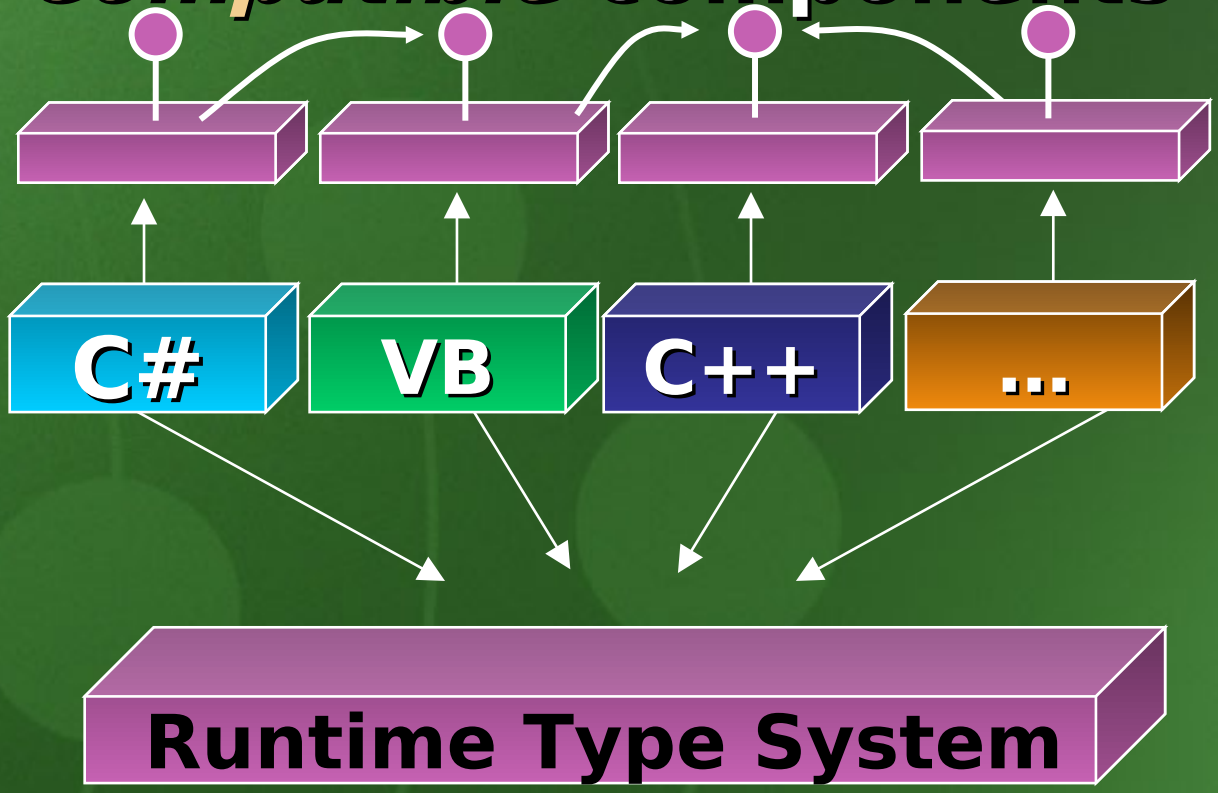


The COM Binary Standard

- Provided language neutrality
- Secondary type system
  - Differed from the type system of the language being used
  - Additional types system to learn
  - Required data transformation

Runtime Type System

Compilers use the Runtime Type System to produce *Type Compatible* components

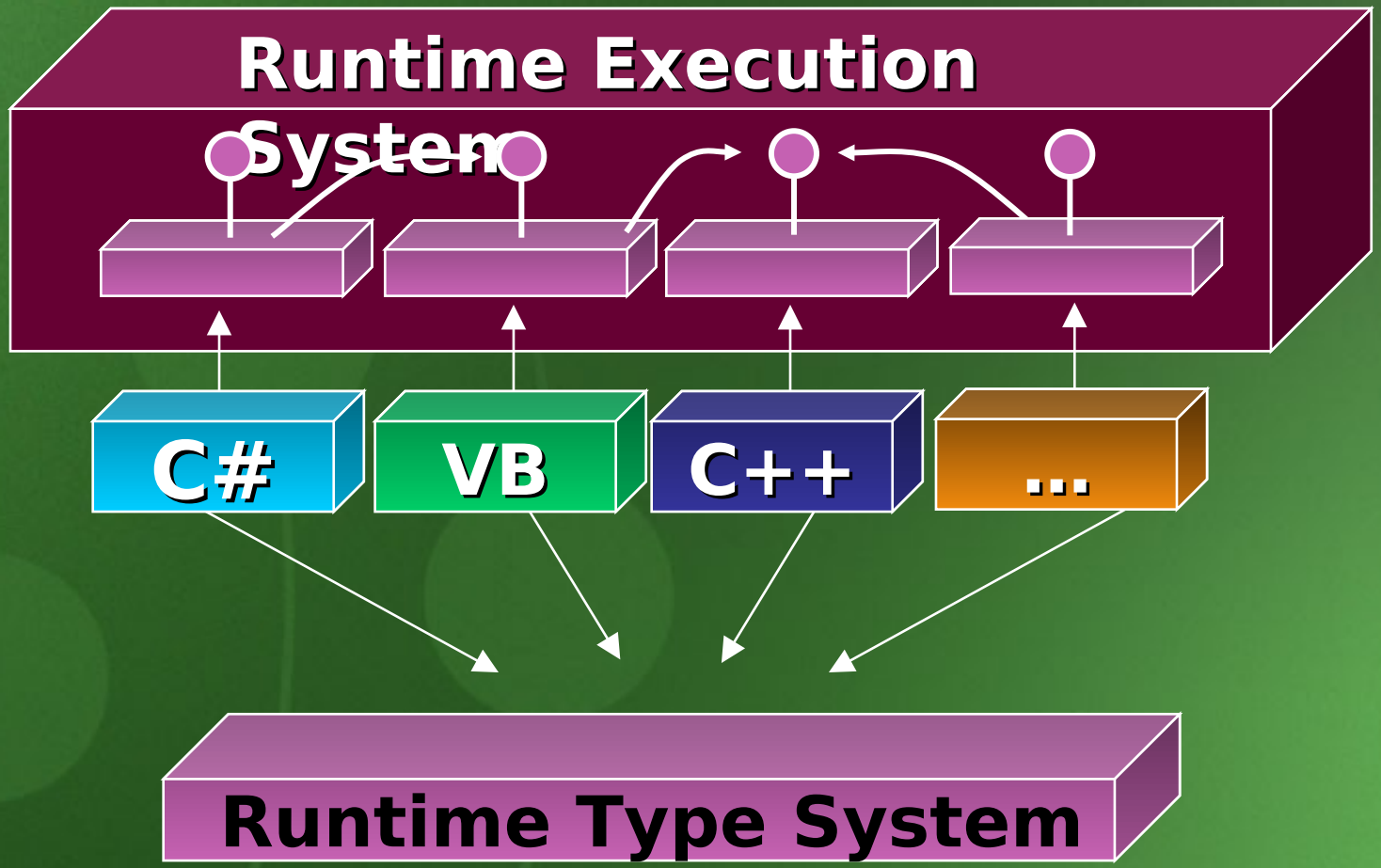


Runtime Components

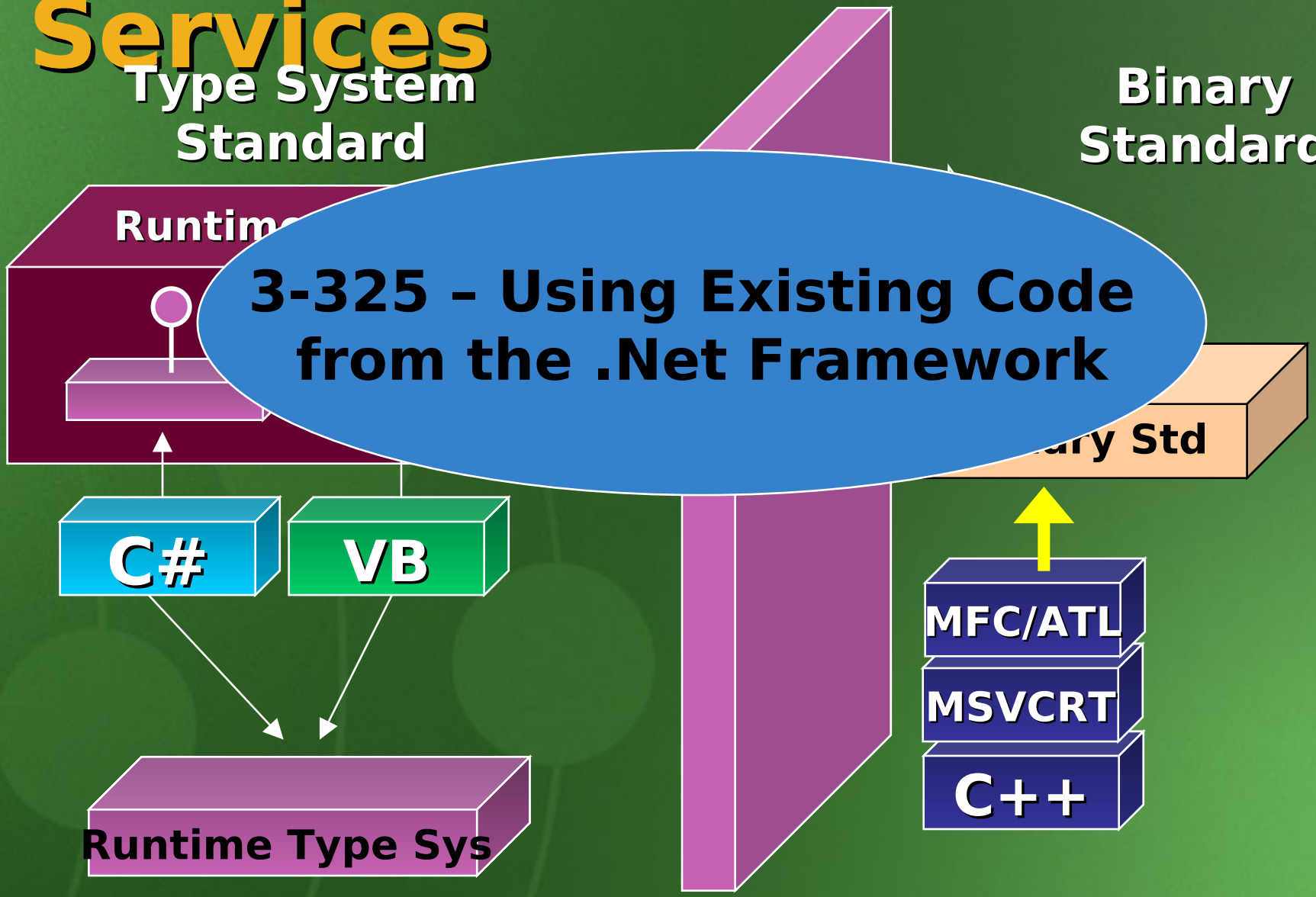
Platform Compilers

Runtime Execution System

Execution System provides additional services for Runtime Types



Interoperability Services



Services That Simplify Development

- Self Describing Objects
  - Automatic Activation
  - Dynamic Inspection
  - Type coercion
- Object Management
  - Activation
  - Type Coercion
  - Error Handling
  - Lifetime management
- Versioning
  - Application Isolation
  - Dynamic Binding
  - Side by Side

Self Describing Objects

- All objects are described in metadata
  - Persisted representation of the type system
  - Stored in executable along with code
  - Produced from type defined in source
  - Emitted directly by compilers
  - Extensible through custom attributes
- Metadata is the single source of type information
  - Consolidates header files, idl, tlbs
  - Consumed by other compilers
  - Consumed by the execution system

No IDL, No Separate Type Libraries

Metadata Richness

- Metadata contains
  - Identity Information
    - Version, Culture, Publisher
  - Type definitions
    - Both internal and exported types
    - Classes, Interfaces, structures, enums
    - Methods, properties, fields and events
  - Type references
    - To dependent types defined elsewhere
    - Versions of dependent types

ncludes assembly version dependencies

Dynamic Inspection

- Frameworks Reflection classes
  - Dynamic access to metadata
  - Inspect types, methods, parameters, etc
  - Create objects
  - Invoke members
  - Invoke methods, properties, events
  - No additional work for class authors
  - Reflection Emit classes

No IDispatch

Assemblies and Type Identity

- Types are packaged, deployed and versioned in an unit called an assembly
- Types are uniquely identified by assembly and type name
- All type references are made through the assembly
- Assemblies can be signed by their publisher
  - Provides name uniqueness
  - Allows secure binding
  - Protects namespace

No GUIDs, CLSIDs or IID's

Application Isolation

- Code sharing is explicit rather than implicit
  - Applications can ensure isolation
    - Unaffected by adding/removing other applications
  - Assemblies can be private or shared
    - Private – xcopy Installed in app directory
    - Shared - Installed in Global Assembly Store

No Registration, No “DLL Hell”, XCOPY install

Metadata Code Walk Thru

Activation

- Classes activated (instantiated) by the common language runtime
  - As opposed to user implemented class factory
  - Made possible by self describing objects
  - Using familiar language syntax (just use new)
  - Past tense, not to be repeated

No Class Factories, No CoCreateInstance,
No DLL entry points, No Registration

Type Coercion

- Types are coerced from one type
  to another through casting
- No need to call QueryInterface
  - Cast is done dynamically without calling into object
- Metadata provides information to determine allowable casts

No QueryInterface to call or implement

Error Handling

- Common exception model used across languages
  - Uses familiar language syntax for handling exceptions
  - Exceptions flow from language to language
  - Integrated with SEH Exception handling
- Use predefined Exceptions classes or create your own

No Hresults, No retval parameters

Lifetime Management

- Lifetime of all managed objects is controlled by the runtime
  - Object is freed when no longer referenced
  - Data moved/compacted to optimize working set
  - No more intrusive than a page fault
- Object reference are automatically traced
  - Even across languages
- Eliminates memory leaks and breaks cycles
  - Leaks cause by incorrect reference

No IUnknown, No ref-counting, No Leaks

Activation Code Walk Thru

Dynamic Binding

- Method calls are bound at runtime
  - Based on method name and signature
  - Converted to v-table call after first use
  - Performance of early binding
  - Resilience of late binding
- Allows method overloading
- Types can evolve over time
  - No dependence on fix object

No Fixed Layout, No Dispatch

Side By Side

- Assemblies form a unit of versioning
- Multiple versions of the same assembly can be installed simultaneously
- References across assemblies can specify version requirements
  - Use current
  - Use specific version
  - Use version greater than x.x
- Binding policy is enforced by the runtime
- Binding policy can be altered by an

Delegate Based Events

- Event support built into the type system
  - First class members like methods and properties
  - Strongly typed like properties
- Tightly coupled
- Cleanly integrated in languages
- Simple user model

No Connection Points

Evidence Based Security

- Permissions granted based evidence obtained by the security system
- Evidence can consists of
  - Zone, publisher, application, assembly, etc
- Each piece of code on the call stack has evidence
- Permission checks can be declarative of imperative
- Access to resource protected with permission checks

Allows Semi-Trusted Execution

Type Safety

- The runtime supports type safe execution
  - Basis for safe execution of downloaded code
  - Ensures that the code is used as it was intended
  - Ensures that only legal operations are performed
  - Prohibits type unsafe operations
- Type safety can be verified at runtime
  - Trust decisions can be based on type safety

More reliable code, safe download

Summary

- Common Type System
  - Tightly Integrated into languages
  - Provides type compatible components
  - Key to simplifying development
    - Eliminates IUnknown, IDispatch, Type Libs, Class Factories, Guids, more
- Common Language Runtime
  - Implementation inheritance
  - Activation
  - Lifetime Management
  - Evidence based security

Related Sessions And References

- 3-332 - .Net Framework Performance Considerations
- 3-325 - Using existing code in the .NET Framework
- 3-331 - Using Transactions and Services in the .Net Framework
- 3-215 - Deploying and Versioning Your .Net Service
- 3-334 - .Net Framework Security
- 3-433 – How does Hello World really Work

Where do you want to go today?

Microsoft